UNIVERSITE MOHAMMED V ***Année 2014- 2015***

Faculté des Sciences – RABAT ***SMI3 – ALGO.II***

Département d'Informatique ***Evaluation1.***

Corrigé

Exercice 1. (4 pts)

L'algorithme de tri par insertion, vu en cours, peut être écrit sous la forme suivante :

Tri_insertion(T, n)

début

pour i := 2 à n faire

insérer(T, i) ;

fpour

fin

(L'algorithme insérer(T, i) dénote ce que fait l'algorithme Tri_insertion à l'itération i.)

1) Décrire, en quelques phrases, l'algorithme insérer(T, i).
   On insère T[i], à sa place, dans le sous-tableau trié T[1..i-1] en le comparant à T[i-1], puis à T[i-2],…jusqu'à ce qu'on rencontre le premier élément T[j] ≤ T[i] (1≤j≤i-1) ou bien T[i] est plus petit que les éléments de T[1..i-1] (dans ce dernier cas j=0) ; à chaque comparaison on décale l'élément qui est > T[i] d'une position à droite. A la fin on insère T[i] à la position j+1.

2) Ecrire le pseudo-code de « **insérer(T, i)** ».

```
début
   clé := T[i] ;
   j := i-1 ;
   tantque (j ≥ 1)  et  (T[j] > clé) faire
         T[j+1] := T[j] ;
         j := j-1 ;
   ftantque
   T[j+1] := clé ;
fin
```

3) Quelle est sa complexité dans le pire des cas ? Rép : O(i) (ou O(i-1))

Le nombre maximum d'itérations de la boucle tant que est i-1. Il y a, au maximum, i-1 comparaisons des éléments de T et i-1 décalages (comparaison et décalage sont les opérations les plus dominantes dans cet algorithme).

On en déduit la complexité du tri par insertion :

$$\sum_{i=2}^{n} O(i) = O(\sum_{i=2}^{n} i) = O(n^2)$$

Exercice 2. (**4 pts**)

On considère l'algorithme suivant :

```
A(n, b)
 // b et n sont des entiers strictement positifs
début
   cpt := 0 ; m := 1 ;
   tantque n ≥ m faire
            cpt := cpt + 1 ;
            m := m * b ;
   ftantque
 retourner(cpt) ;
fin
```

i) Faites tourner l'algorithme pour n =15 et b = 2. Quelle est la valeur de A(15, 2) ?
Rép : 4

ii) Quel est le nombre d'exécutions du corps de la boucle tantque (ou le nombre d'itérations)? Rép : $1 + E(\log_b n)$. Justifiez votre réponse.
Si on désigne par cpt_i, m_i les valeurs respectives de cpt et de m à l'itération i de tantque, on peut montrer (par récurrence sur i) que $cpt_i = i$ et $m_i = b^i$.(i = 0 correspond à l'initialisation de cpt et m.)
Avec cette notation, la boucle tantque peut être interprétée comme suit :

```
  i := 0 ;
Tantque n ≥ m_i faire
         i := i+1 ;
         cpt_i := cpt_i-1 + 1 ;
           m_i := m_i-1 * b ;
ftantque
```

Soit p le nombre d'exécutions de la boucle tant que (p constitue la dernière exécution du corps de tantque). p vérifie :

1) $n \geq m_{p-1}$ (condition de tantque pour faire la dernière exécution numéro p) et
2) $m_p = m_{p-1} * b$ (exécution du corps de tantque à l'itération p) et
3) $n < m_p$ (car p est la dernière exécution)

On en déduit que :

$$b^{p-1} \leq n < b^p \Rightarrow p-1 \leq \log_b n < p \Rightarrow p-1 = E(\log_b n)$$

D'où $p = 1 + E(\log_b n)$ (avec $\log_b n = Ln(n) / Ln(b)$)

Remarque : pour n = 15 et b = 2 on a : $2^3 \leq 15 < 2^4 \Rightarrow E(\log_2 15) = 3$

iii) Que représente la valeur retournée par cet algorithme dans le cas où b = 2 ?
L'algorithme calcule le nombre de chiffres binaires de n en base 2 ($15 = (1111)_2$).
(ou le nbre de bits nécessaires pour représenter n (il peut être obtenu en prenant les restes de la division successive de n par 2 (voir exemple du chap1))

Exercice 3. (4 pts)

Remplir le tableau suivant, ligne par ligne, en écrivant dans chaque case le symbole de complexité asymptotique le plus adéquat. Le symbole X ($X \in \{\Omega, \Theta, O\}$), dans une case de ligne f et de colonne g, est interprété comme f = X(g) et non pas comme g = X(f).

	n	$\sqrt{n} \log n$	n^2	2^{1000}
n	Θ	Ω	O	Ω
$\sqrt{n} \log n$	O	Θ	O	Ω
n^2	Ω	Ω	Θ	Ω
2^{1000}	O	O	O	Θ

Le remplissage est basé sur les remarques suivantes :

1- $\log n \le n^a \quad \forall n \ge 1, \ \forall a > 0$; en particulier pour a = 0.5, on a : $n^{0.5} \log n \le n$ (TD2)
2- $f = \Theta(f)$ ($f = O(f)$ et $f = \Omega(f)$)
3- $f = O(g) \Leftrightarrow g = \Omega(f)$
4- $2^{1000} = O(1)$
5- $O(1) \subset O(\sqrt{n} \log n) \subset O(n) \subset O(n^2)$

Exercice 4. (8 pts)

Etant donné un tableau A[1..n] de n entiers. Les éléments de A sont pris dans l'ensemble {1,2,…n}, i.e. $\forall i \in \{1,2,..,n\}$ $1 \le A[i] \le n$. Un élément de A peut se répéter plusieurs fois dans le tableau A. Le nombre de répétions d'un élément est appelé fréquence.

exemple :

A

5	2	1	5	1	9	10	9	1	5
1	2	3	4	5	6	7	8	9	10

1) Ecrire une fonction « **fréquence(x, A, i, j)** », qui retourne la fréquence d'un entier x dans le sous-tableau A[i..j] ($1 \le i \le j \le n$). Quel est le nombre de comparaisons de x ? (par exemple la fréquence de 1, dans A[3..6], est 2)
On parcourt le sous-tableau de i à j et on compte le nombre d'occurrences de x.

```
Fonction fréquence(x : entier, A : tableau[1..100] de entier, i : entier, j : entier) : entier
cpt, k : entier ;
début
   cpt := 0 ;
   pour  k := i  à  j  faire
          si (A[k]  = x)  alors  cpt := cpt + 1 ;
          fsi
   fpour
retourner (cpt)
fin
```

Le nombre de comparaisons de x, dans la fonction fréquence, est égal à j – i +1.

2) a) Utiliser cette fonction pour écrire un algorithme (ou une fonction) « **affiche_fr(A, n)** », qui affiche chaque élément de A suivi de sa fréquence, (Les éléments de A sont affichés sans répétition).

(exemple d'affichage :   5 : 3   2 : 1   1 : 3   9 : 2   10 : 1)

L'algorithme affiche-fr(A,n) a la forme suivante :

```
pour  i := 1  à  n  faire
   si A[i] n'a pas été rencontré auparavant alors
         écrire(A[i]) ; écrire(fréquence(A[i], A, i, n))
    // l'appel à fréquence retourne la fréquence de A[i] dans le sous-tableau A[i..n]
//car A[i] n'a pas été rencontré dans le sous-tableau A[1..i-1](A[i] n'est pas dans
//le ss-tableau A[1..i-1], inutile de le chercher dans ce ss-tableau)
   fsi
fpour
```

A[i] a été rencontré auparavant ⇔ ∃ k ∈ [1..i-1] t.q. A[i] = A[k] ⇔ faire une recherche de A[i] dans le sous-tableau A[1..i-1], et donc i-1 comparaisons de plus pour chaque i (au total on fait $O(n^2)$ comparaisons de plus).

Par hypothèse les éléments de A sont dans l'ensemble {1,…,n}, donc ils peuvent être utilisés comme indice d'un tableau (de booléen par exemple, déjà vu en TD1).On utilise un tableau de booléens traité[1..n], initialisé à faux, tel que :

Traité[k] = vrai signifie que k est un élément du tableau et k a été rencontré auparavant. (Traité[k] = faux signifie que k n'est pas un élément du tableau ou bien k est un élément du tableau qui n'a pas été rencontré auparavant (autrement dit : k n'est pas traité)).

Dans l'exemple précédant, lorsqu'on traite A[1] on marque traité[5]= vrai pour ne pas traiter prochainement A[4] et A[10]   (traité[A[4]] = traité[A[10]] = vrai).

```
Algorithme  affiche_fr(A, n)
début
   pour  i := 1  à  n  faire traité[i] := faux ;
   fpour
   pour  i := 1  à  n  faire
         k := A[i] ;
         si  (traité[k] = faux)  alors
            écrire(k) ; écrire(fréquence(k, A, i, n)) ;
            traité[k] := vrai ;
         fsi
   fpour
fin
```

b) Quel est le nombre total de comparaisons des éléments de A ? Donner un ordre de grandeur de ce nombre (utiliser grand-O).
(La note maximale de cette question est attribuée à celui qui utilise le moins de comparaisons possible.)

Le nombre total de comparaisons dépend des appels à la fonction fréquence.

Soit C(i,j) le nombre de comparaisons de la fonction fréquence pour calculer la fréquence d'un élément dans le sous-tableau A[i..j]. $C(i,j) = j-i+1$.

Si on note par T(n) le nombre de comparaisons de l'algorithme affiche_fr dans le pire des cas, alors

$$T(n) = \sum_{i=1}^{n} C(i,n) = \sum_{i=1}^{n} (n-i+1) = n^2 - \frac{n(n+1)}{2} + n = \frac{n(n+1)}{2}$$

Remarque : Le pire des cas correspond à la situation où les éléments de A sont tous distincts i.e. la fréquence de chaque élément de A est égale à 1 ; autrement dit A est une permutation de S_n.

$T(n) = O(n^2)$.